Otimização de Scraping de Páginas Dinamicas Utilizando Grafos

Alan da Mota Nascimento[1], **Gabriel Alberto Moura de Sá**[1]

[1] Instituto de Educação Superior de Brasília (IESB)

gabrielsa08@gmail.com, alan.nascimento@iesb.edu.br

Abstract.

Resumo. *O uso de Web Scrapers é uma famosa alternativa para extrair dados de forma automática de websites. Ao programar um, uma das primeiras etapas é descobrir se o site a ser "garimpado" é estático ou dinamico. Websites estáticos podem ser processados rapidamente com uma requisição, já websites dinâmicos precisam esperar pela execução de seu JavaScript antes de se obter o HTML final, impondo uma etapa a mais. Para explorar o tempo gasto no Scraping de páginas dinamicas e como otimiza-lo, neste trabalho será desenvolvido uma plataforma que, utilizando grafos, consiga traçar a rota com o menor custo dado um conjunto de destinos fornecidos pelo usuário. Os valores que alimentarão este grafo ponderado serão obtidos através dos preços de passagem provenientes do Scraping de uma página dinâmica.*

1. Introdução

Atualmente, ao criar um pacote de viagens, isto é, definir diversos destinos que serão visitados de avião, é necessário pesquisar o preço das passagens individualmente até chegar a um resultado satisfatório. Felizmente, este problema pode ser solucionado de forma totalmente automatica utilizando um web scrapper e algoritmos que permitam encontrar o caminho com menor custo.

Para chegar numa solução viável, diversas variáveis devem ser consideradas. Para começar, o uso de um scrapper não é a solução ideal, posto que será necessário diversas consultas para obter os dados necessários. Levando este problema em consideração, o uso de uma API própria para isto, ou seja, capaz de retornar os valores das passagens desejadas seria mais eficiente. Mesmo assim, será feito um scrapper para realizar o garimpo dos dados, posto que não existem opções gratuitas de uma API que faça isto, diminuindo sua acessibilidade.

Dado o exposto, neste artigo será definido melhor os problemas relacionados ao uso do scrapper e do problema de menor caminho, buscando encontrar soluções viáveis levando em consideração o tempo gasto em cada consulta.

2. Objetivos Gerais

O objetivo neste artigo é obter uma solução viável para o Scraping de múltiplas páginas Web dinâmicas, tendo como principal métrica o tempo gasto do momento em que o usuário define o pacote de viagens até o final do cálculo do caminho de custo minimo.

3. Metodologia

3.1. Escolhendo o site para fazer o Scraping

O site escolhido para fazer os scraps foi o ”https://www.decolar.com.br”. Para chegar no site escolhido, diversos fatores foram levados em consideração. O site é confiável e retorna diversos preços de passagens para cada consulta. Além disso, também tem links estáticos e é estruturado de forma que os dados importantes são coletados facilmente.

3.2. Definição do problema

Partindo do contexto de que será montado um grafo ponderado, a primeira etapa é a definição dos vértices. Para fazer isso, será solicitado ao usuário que informe destinos distintos que formarão um pacote, estes destinos vão ser os vértices do grafo. Levando em consideração que o preço das passagens variam de acordo com a data, também é necessário saber qual será a data de partida, ou seja, em que dia o usuário pretende partir do destino inicial (Chamaremos de origem), e qual o tempo em dias que ele pretende ficar em cada destino.

Utilizando os dados informados acima, é possível definir um grafo contendo N vértices, e sabendo que todos os vértices são adjacentes, é formado um grafo completo, isto é, um grafo simples em que todos os vértices possuem uma aresta conectando-os.

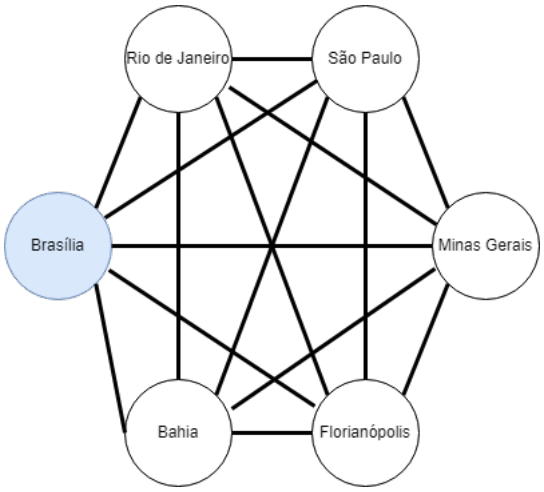

Figure 1. Definição do grafo completo contendo cinco destinos e uma origem, destacada com a cor azul.

Após definir o grafo, resta encontrar os valores de cada aresta. Esta etapa envolve o scrapping dos preços das passagens. É inviável realizar todas as consultas de uma vez, posto que estes dados são coletados numa página dinâmica e os valores das arestas variam de acordo com a data, isto é, num dia específico cada aresta teria um valor X, mas em outro dia este valor não seria o mesmo. Levando em consideração um grafo contendo N vértices, para conseguir todos os valores possíveis de cada aresta seriam necessários $\frac{N(N-1)}{2} * N$ consultas, sendo este o principal gargalo da aplicação.

3.3. Problema do Caixeiro Viajante

Ao definir o problema, torna-se evidente que este se enquadra no problema do caixeiro viajante. Em síntese, o usuário deverá receber qual vai ser o caminho que o permitirá visitar todos os destinos uma única vez e voltar a origem com o menor preço.

Com isto, é necessário trazer algumas limitações para o usuário. Como o problema do caixeiro viajante é um problema de complexidade NP-Díficil, limitar a quantidade de destinos para seis incluindo a origem evitará que sejam feitas consultas que tomarão muito tempo. Após isso, é ncessário encontrar um algoritmo que encontre o menor caminho com o mínimo de consultas possível, pois como explicado na subseção anterior, o custo dos vértices é uma função que depende também do tempo, tornando a complexidade do problema ainda maior [Boulic et al. 2008].

3.4. Solução Naive

Uma possível solução para o problema acima envolve realizar o trajeto de trás para frente, isto é, será consultado os preços de todos os destinos para o destino final (Que também é a origem), e então o vértice que possui a aresta com o menor custo destes será utilizado como o destino final para a próxima consulta. Este processo se repetirá até que sobre apenas um destino, finalizando com uma última consulta da origem até este destino.

Esta solução reduz a quantidade de consultas necessária já que é feito de forma dinâmica, permitindo manter as arestas que não estão ligadas diretamente ao destino final sem valor, mas não vem sem desvantagens, afinal, o resultado não necessariamente será de fato o caminho com menor custo.

4. Resultados obtidos

Para a tabela abaixo, foram realizadas 20 consultas com 3, 4, 5 e 6 destinos, e então calculado a média de tempo gasto em cada.

	Total destinos			
	3	4	5	6
Naive	40.55s	75.37s	110.06s	150.20s
Naive [Headless]	38.32s	68.88s	103.89s	147.98s

Inicialmente, é possível inferir pelo algoritmo naive que o uso do headless (Opção do scrapper de ser executado no modo invísivel) já reduz o tempo médio. Tal otimização mostrou-se constante nos testes, consequentemente será a padrão para a próxima tabela. Vale ressaltar que qualquer otimização ao scrapper é indispensável, posto que o mesmo é o principal gargalo.

5. References

References

Boulic, R., Gamache, M., and Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. In Megiddo, N., editor, *Discrete Optimization*.